QML for Desktop Applications

Helmut Sedding

Michael T. Wagner

IPO.Plan GmbH

Qt Developer Days Berlin 2012

About us

- IPO.Plan GmbH
- Located in Ulm and in Leonberg near Stuttgart
- The company is both experienced in factory planning and in software development.
- Our software focuses on process and logistics planning

QML for Desktop Applications

- Real World Usage: IPO.Log
- Tight Data Coupling
- QML for 2D Editing
- Desktop GUI
- Résumé

Real World Usage

- IPO.Log is used by manufacturing industries for assembly process and logistics planning
- IPO.Log provides a GUI tailored to its specific needs
- To allow for a modern, streamlined GUI and rapid development we chose QML
- QML brings the highly customized graphical Web & Mobile User Interfaces to the desktop
- www.ipolog.de

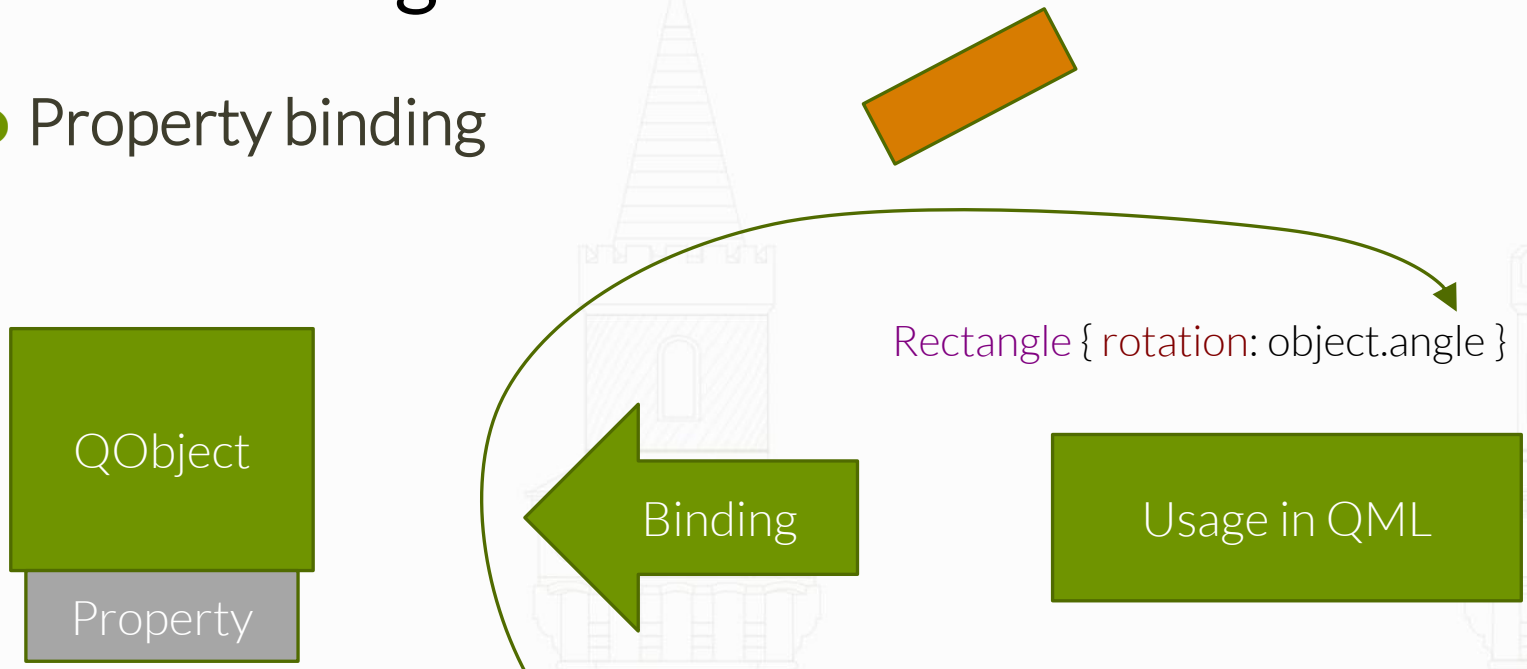
Tight Data Coupling: Values

Connect C++ Data Models to QML Views

Helmut Sedding and Michael T. Wagner | IPO.Plan

Data Binding

- Property binding



Q_PROPERTY(qreal angle READ angle WRITE setAngle NOTIFY angleChanged);

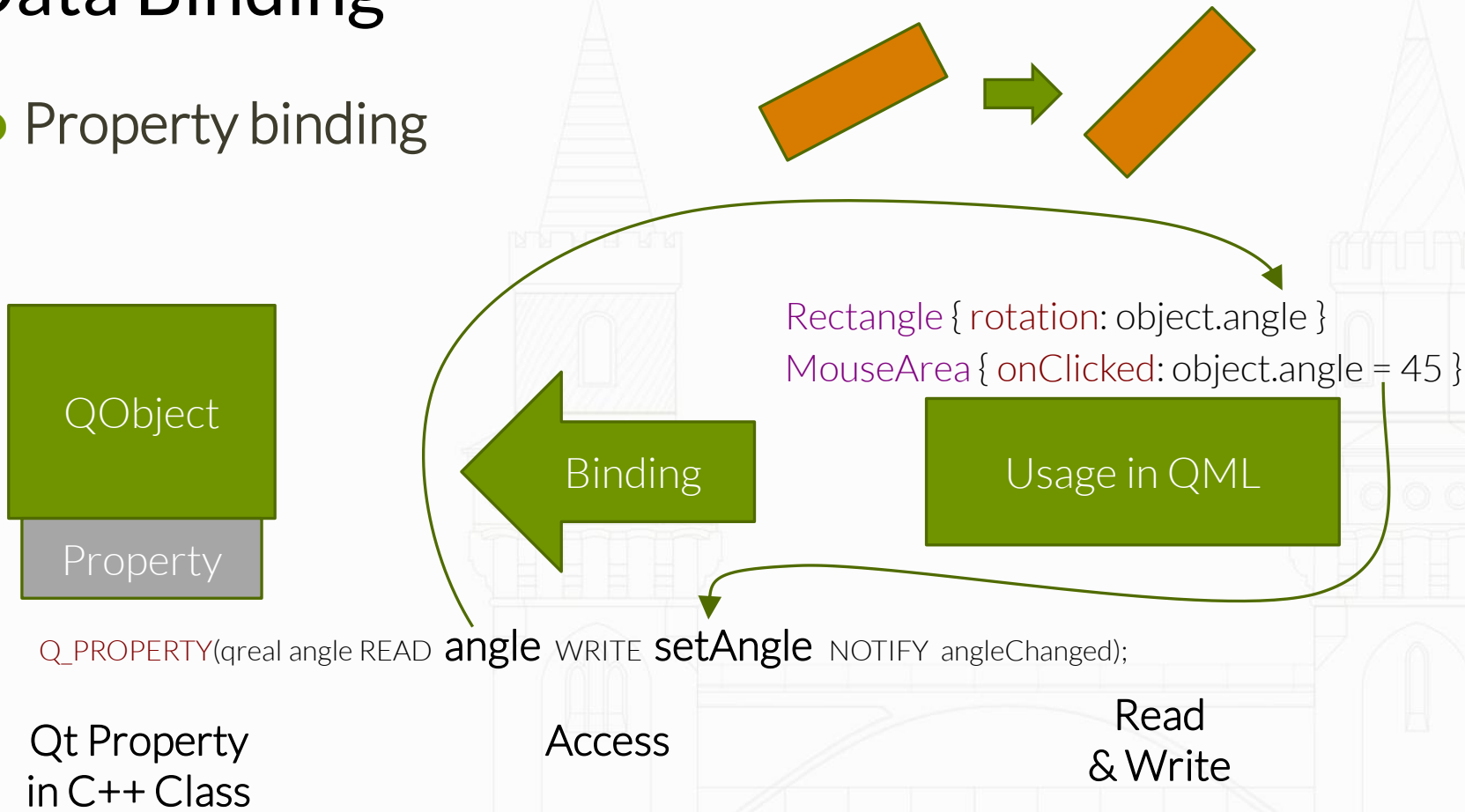
Qt Property in C++ Class

Access

Read & Write

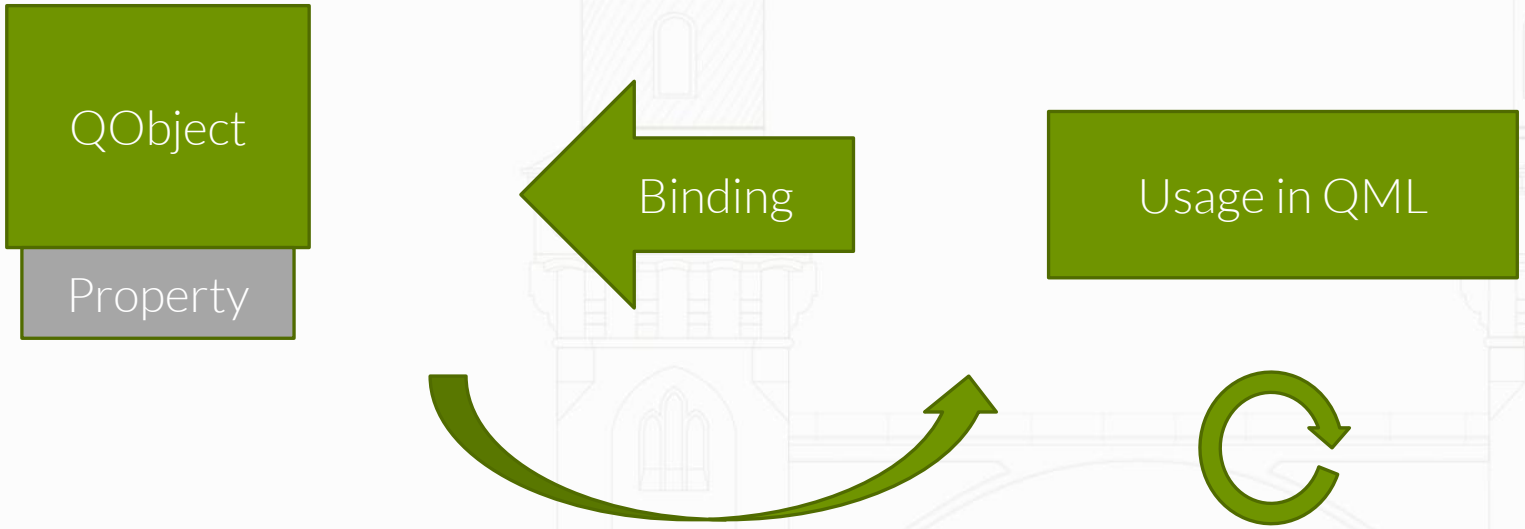
Data Binding

- Property binding



Data Binding

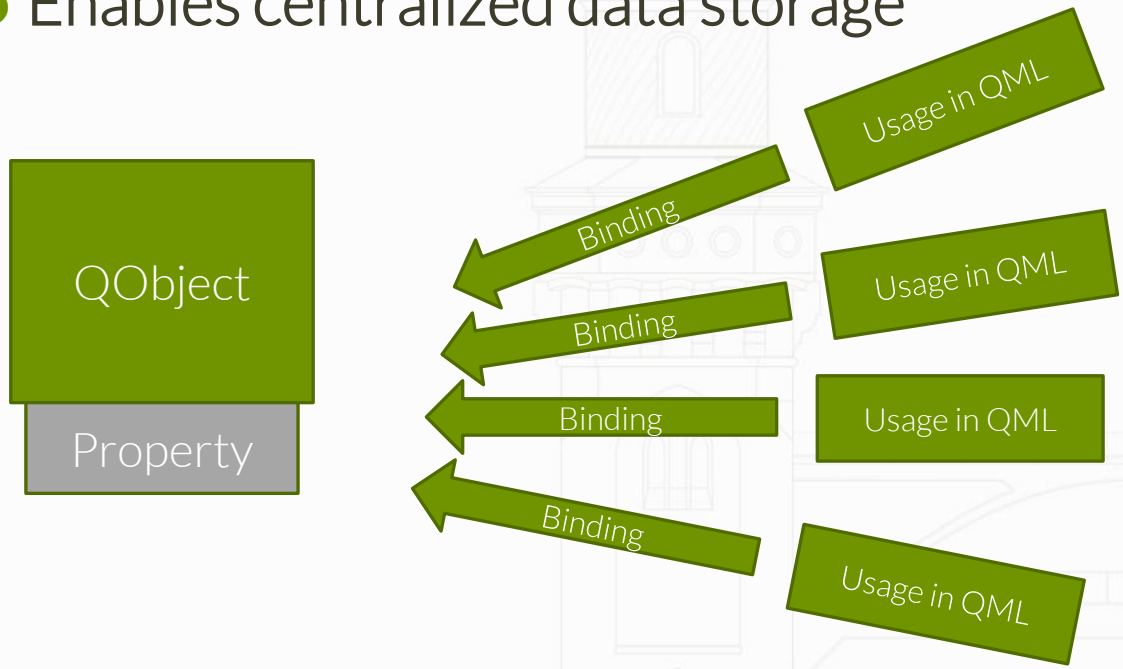
- Property binding
- Change propagation via Notification signals



On Change: 1. Change Notification 2. Recalculation

Data Binding

- Property binding
- Change propagation via Notification signals
- Enables centralized data storage



Data Binding

- Property binding
- Change propagation via Notification signals
- Enables centralized data storage

- Advantages:
  - Subscription based model views
  - Q_PROPERTY macros define clear interface
- Disadvantages:
  - Signal setup for each binding: 50% slower than const values
  - On Notify: update time scales linear with usages

Selection: Example for slow data binding

- Display a list of numbers
- Task: display “SEL” at selected index, else “---”

```
property int selectedIndex: 7
```

0	---
1	---
2	---
3	---
4	---
5	---
6	---
7	**SEL**
8	---
9	---

Selection: Example for slow data binding

- Display a list of numbers
- Task: display “SEL” at selected index, else “---”

```
property int selectedIndex: 7
```

```
4
```

0	---	---
1	---	---
2	---	---
3	---	---
4	---	**SEL**
5	---	---
6	---	---
7	**SEL**	---
8	---	---
9	---	---

Selection: Example for slow data binding

- Display a list of numbers
- Task: display "SEL" at selected index, else "---"

```
property int selectedIndex: 7
```

4

0	---	---
1	---	---
2	---	---
3	---	---
4	---	**SEL**
5	---	---
6	---	---
7	**SEL**	---
8	---	---
9	---	---

MouseClick

Selection "naïve": notifications costly

- Task: display "SEL" at selected index, else "---"
- Naïve Solution:

```
property int selectedIndex: -1
Column {
    id: rep
    Repeater {
        model: 1000
        delegate: Text {
            property bool isSelected: index == selectedIndex
            text: isSelected ? "SEL" : "---"
            MouseArea { anchors.fill:parent; onClicked: selectedIndex = index }
        }
    }
}
```

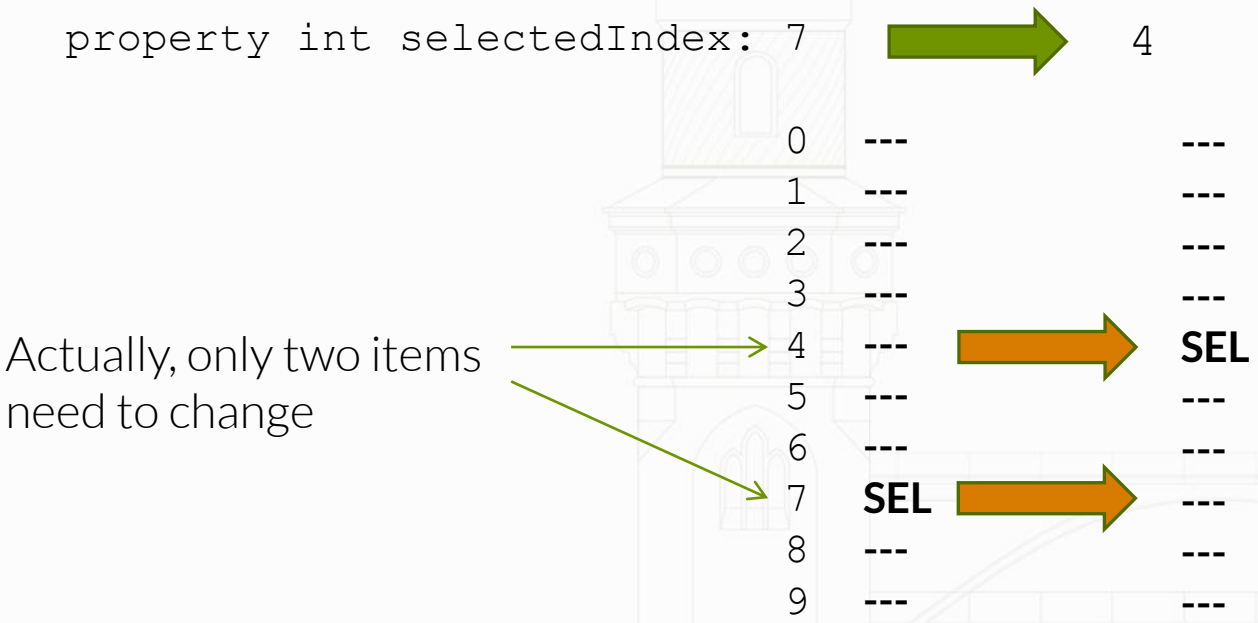
re-evaluates on change

- Slow on change, because *all* delegates are notified
- Insufficient for big applications

* Number of Items

Selection: Example for slow data binding

- Task: display “SEL” at selected index, else “---”



Selection “quick”: update selected item only

- Solution with constant update time:

```
property int selectedIndex: -1
property int selectedIndexBefore: -1
onSelectedIndexChanged: {
    if(selectedIndexBefore>=0) { rep.children[selectedIndexBefore].isSelected = false }
    if(selectedIndex>=0) { rep.children[selectedIndex].isSelected = true }
    selectedIndexBefore = selectedIndex
}
Column {
    id: rep
    Repeater {
        model: 1000
        delegate: Text {
            property bool isSelected: false
            text: isSelected ? "SEL" : "---"
            MouseArea { anchors.fill:parent; onClicked: selectedIndex = index }
        }
    }
}
```

re-evaluates on change

* 2

- Quick on change: *only two* delegates are updated

Selection of QObjects: improved handling

- When using C++ data models
  - Quick selection handling can be provided efficiently by a hard coded *isSelected* property, that is written centrally

```
Q_PROPERTY(bool isSelected READ isSelected NOTIFY isSelectedChanged);
```

- Updates in constant time
- Selection handling happens at one single point only

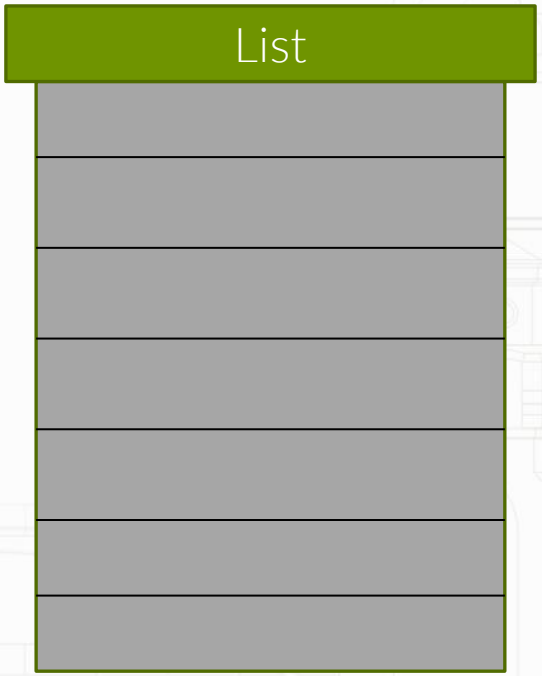
Tight Data Coupling: Lists

Connect C++ Data Models to QML Views

Helmut Sedding and Michael T. Wagner | IPO.Plan

Data Model Requirements

- How can lists of QObject* be efficiently stored in C++, and handled transparently by QML?
- Requirements:
  - Easy and quick C++ handling
  - Detailed Repeater updating
    - On Add/Remove: non-changing items remain
  - Pass List as function parameters



Data Model: Alternatives

- QList<T>, QVariantList
  - No detailed Repeater Updating (only total reset)
- QML ListModel
  - No access from C++
- QAbstractListModel
  - Slow and tedious access in C++ with QVariant, QModelIndex
- QObjectListModel
  - Proposed solution

Data Model: QObjectListModel*

- QObjectListModel*
  - Base class: QAbstractListModel
  - Stores QList< QObject*> internally
  - Sends Add/Remove signals
- Provides solution for both C++ and QML:
  - C++: Accessors typed by QObject* are quick and easy to handle
  - Repeaters can deal with its base class: QAbstractListModel
  - Pointer has small memory footprint in method arguments
- QObjectListModelT<T>*
  - Same as above, but additionally typed

- This way, C++ storage is efficient and transparent for QML

Accessing QObjectListModel items

- Provide Property for QML access:
  - `Q_PROPERTY(QObjectListModel * list READ list CONSTANT);`
- By Integer (array-index):
  - `list.get(i)`
- By Object:
  - `var i = list.indexOf(object)`
- By Name:
  - `var i = list.indexOfName("Crichton")`

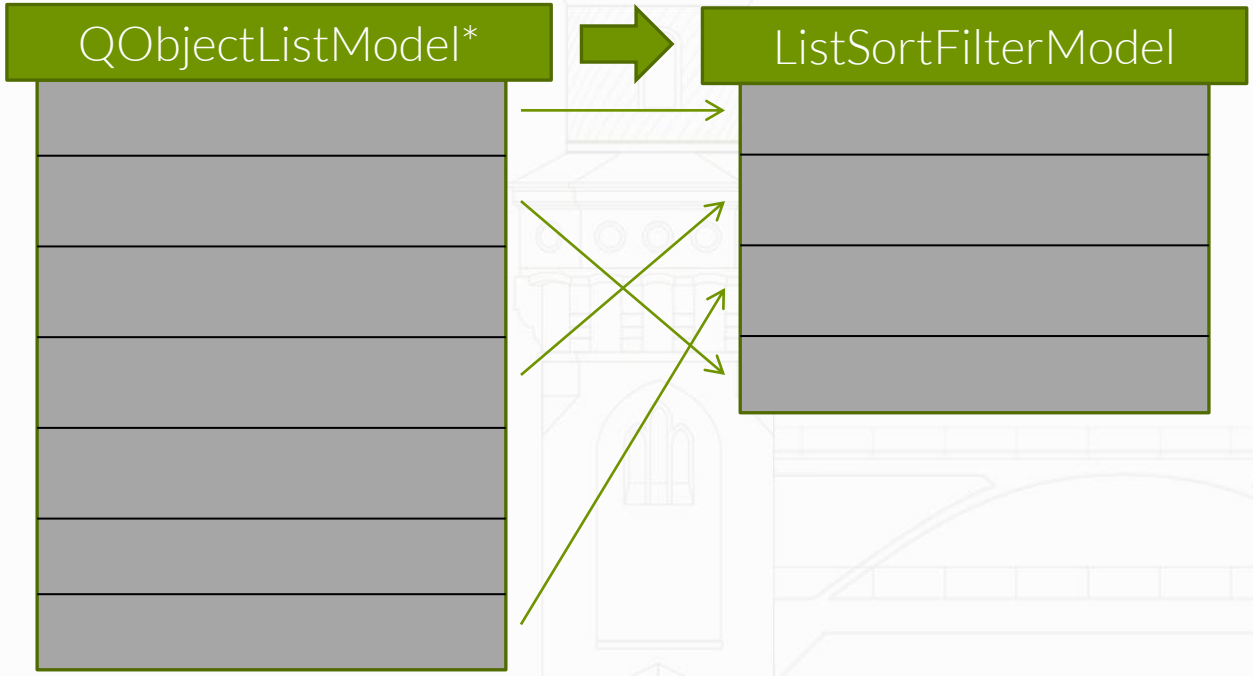
- We extended this to provide constant access time with self-updating index if needed

Typed List: QObjectListModelT<T>*

- Typed QObjectListModels:
  - ```
    class RackListModel : public QObjectListModelT<Rack *> {
    };
    ```
- Statically typed c++ accessors:
  - ```
    Rack * rack = list.at(3);
    ```
- Typed Property for QML access:
  - ```
    Q_PROPERTY(RackListModel * racks READ racks CONSTANT);
    ```
- Beforehand, make the list available in QML:
  - ```
    qmlRegisterUncreatableType<RackListModel>("IpoLog",3,0,"RackListModel",QString());
    ```

Filtering&Sorting QObjectListModels

- Proxy Models can filter or sort other list models.
- Updates are forwarded though proxy models



Filtering&Sorting QObjectListModels

- Proxy Models can **filter** or **sort** data.
- Updates are forwarded though proxy models

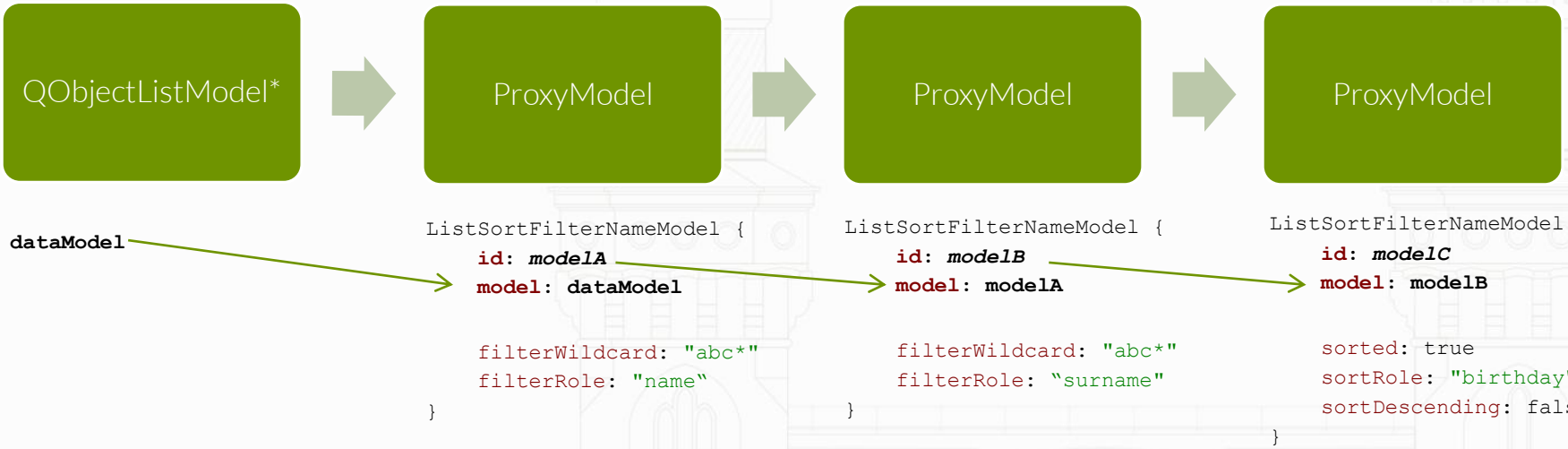
```
ListSortFilterNameModel {
    id: sortFilterModel
    model: dataModel

    filterWildcard: "abc*"
    filterRole: "name"
    filterCaseSensitivity: ListSortFilterNameModel.CaseInsensitive

    sorted: true
    sortRole: "birthday"
    sortDescending: false
}
Repeater {
    model: sortFilterModel
    ...
}
```

INPUT

OUTPUT

Proxy Model Chaining

- Proxy Models can even be chained
- Here e.g. multiple string filters



Customized Proxy Models

- There often arise custom filtering needs:
  - e.g. object.nr < 100

Performance
- suitable for lists with ca. 1000 items.
- If it's not quick enough, simply switch to a C++ proxy model implementation

- Custom filtering achieved by defining javascript methods that are called from C++

```
ListFilterModel {
    model: dataModel
    filtered: true
    function filterAccepts(index, obj) {
        return object.nr < 100
    }
}
```

- Sorting is similar, calling `lessThan`

Tight Data Coupling: Summary

- Property binding and QObjectListModel*
  - allows for centralized data storage
  - Usable both in C++ and QML
  - easy change propagation
  - Careful when using many bindings at the same time
    - Slow setup and teardown

QML for 2D Editing

Viewing and Editing 2D Objects

Helmut Sedding and Michael T. Wagner | IPO.Plan

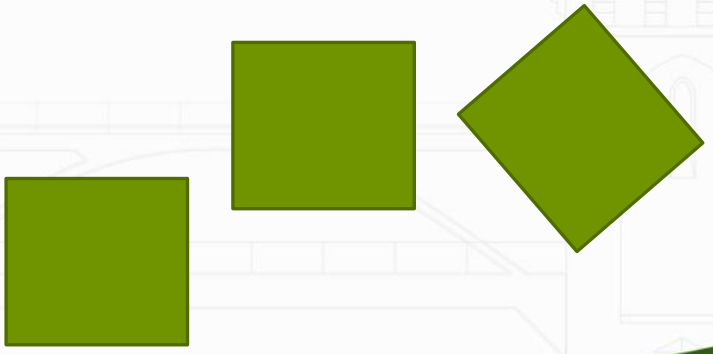
Flickable: A Scrollable 2D Canvas

- Scrolling looks good in QML
- Repeater puts objects into scene
- Objects positioned using data binding
- Polygons drawn by QPainter in QGraphicsItems

Repeater creates objects

- Data Model of geometric objects
- Each object has
  - Transformation
    - *position*
    - *angle*
  - Size
    - *boundsMinimum*
    - *boundsMaximum*

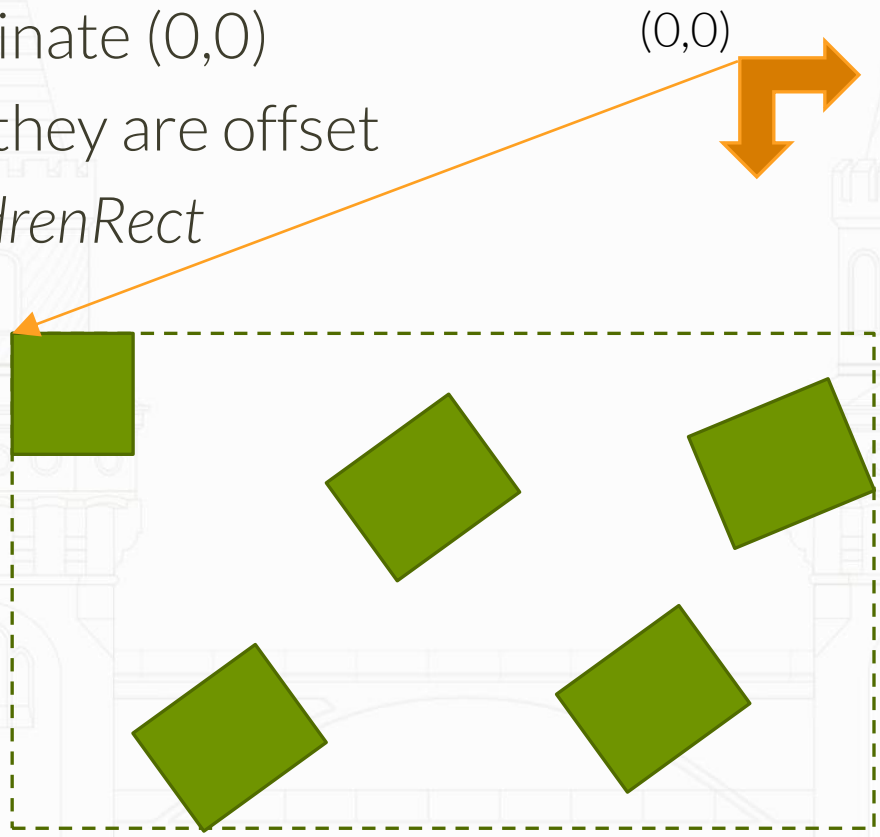
```
Repeater {
    model: workspace.racks
    delegate: Item {
        x:object.position.x
        y:object.position.y
        rotation: object.angle
        Rectangle{
            width:(object.boundsMaximum.x-object.boundsMinimum.x)
            height:(object.boundsMaximum.y-object.boundsMinimum.y)
            color: "#ccc"
        }
    }
}
```



Flickable: Bounding Calculation

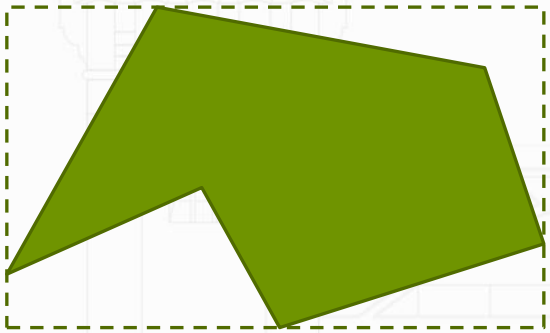
- Flickable starts at coordinate (0,0)
- But items don't do that, they are offset
- Therefore offset by *childrenRect*

```
Flickable {
    id: outer
    contentWidth: inner.width
    contentHeight: inner.height
    Item {
        id: inner
        x: -childrenRect.x+50
        y: -childrenRect.y+50
        width: childrenRect.width+100
        height: childrenRect.height+100

/* CONTENT HERE */

    }
}
```



Polygon: Drawn by Custom QML Item

- Polygons are not supported by QML
- Resorting to QGraphicsItem
  - Which lives perfectly fine in QDeclarativeScenes
  - Drawing with QPainter
  - Non-Rectangular shape requires custom mouse hit testing



Editing For Many Complex Items

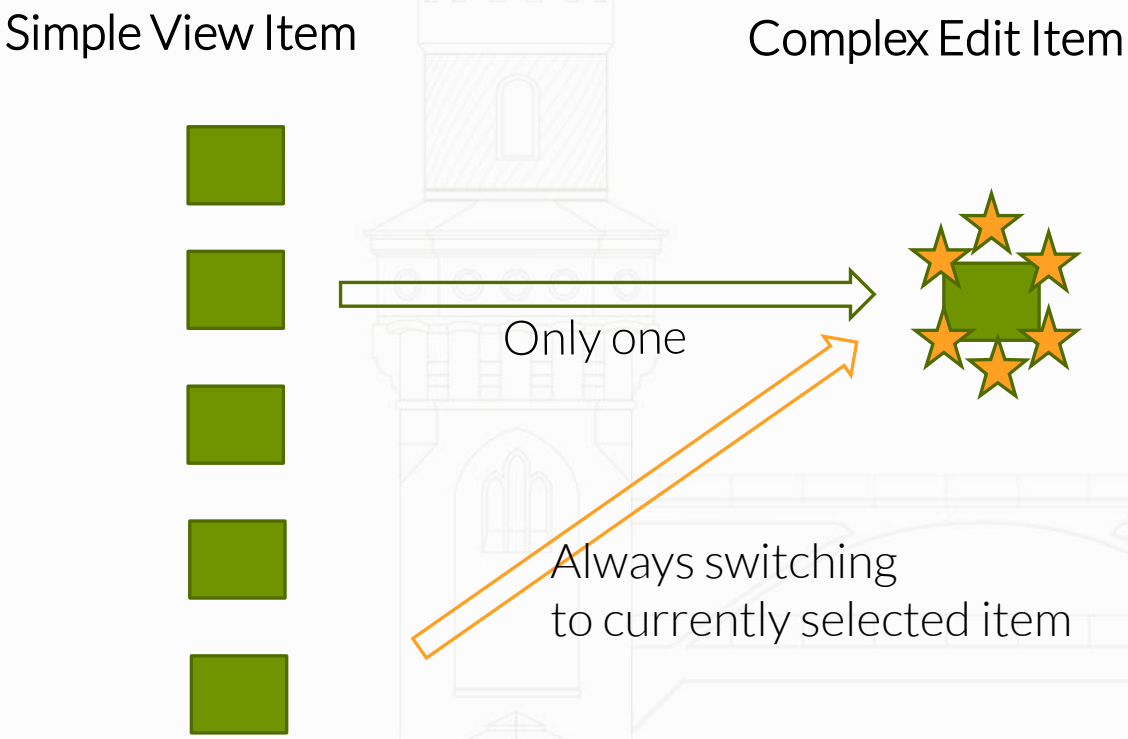
- Naïve Solution: Hide not needed Edit Components
- Drawback: memory requirements and setup/teardown times

Simple View Item

Complex Edit Item

Editing: Single Edit Component

- Save memory by using the Single Edit Component pattern
  - Split view into simple view items and few complex edit items



QML for 2D Editing: Summary

- Flickable works quite well
  - Scrolling
  - Zooming
  - Content Fit
- For Complex Graphic Items
  - use fallback solution: C++ rendering (e.g. for polygons)
  - limit element count, e.g. use the Single Edit Component pattern
- Next improvements
  - Level of Detail
  - Lazy loading
- Limitations
  - Flickable redrawing is not perfect

Desktop GUI

Viewing and Editing 2D Objects

Helmut Sedding and Michael T. Wagner | IPO.Plan

Tool Tips

- Defined easily:

```
ImageButton {
    text: "Do"
    ToolTip.text: "Does nothing"
}
```

Tool Tips

- Defined easily:

```
ImageButton {
    text: "Do"
    ToolTip.text: "Does nothing"
}
```

- Implemented as an attached property:

```
class ToolTipAttached : public QObject
{
        Q_OBJECT;
        Q_PROPERTY(QString text READ text WRITE setText NOTIFY textChanged);
public:
        static ToolTipAttached *qmlAttachedProperties(QObject *obj);

        ToolTipAttached(QObject *parent) : QObject(parent) {}
 …
};
QML_DECLARE_TYPEINFO(ToolTipAttached, QML_HAS_ATTACHED_PROPERTIES)
```

Drag-n-Drop

- Custom DragArea, DropArea items
- Using standard Qt Drag-n-Drop implementation

```
DragArea {
    enabled:avoDragEnabled
    anchors.fill: parent
    supportedActions: Qt.MoveAction
    data {
        text: "Process"
        source: parent
    }
    onDragStart: {}
    onDragEnd: {}
}
```

```
DropArea {
    anchors.fill: parent
    onDragEnter: {}
    onDragLeave: {}
    onDrop: {
        event.accept(Qt.MoveAction);
        doDrag(event.data.source);
    }
}
```

Desktop GUI: Limitations

- Custom QML Items are handy but not always
  - Too many cases make abstraction slow
    - When e.g. Button.qml both supports Image and Text
  - Rather come up with more specialized items
    - e.g. TextButton.qml and ImageButton.qml
- Mouse Input is sufficient for desktop use
  - But we did not need context menus
- Keyboard input is tedious:
  - tab orders, shortcut keys
- ListViews and Scrollbars don't fit together well
  - Delegate item height can't be fixed

Résumé

QML makes desktop GUIs attractive again

Helmut Sedding and Michael T. Wagner | IPO.Plan

Advantages

- Animations look stunning and are easy to create
- Easy to change without recompiling
- Pixel-perfect UI is created quickly
- Data-Binding simplifies update-routines

Disadvantages

- Display of many elements requires fine-tuning
  - Fallback to fast C++ QGraphicItems is possible
- Keyboard input is tedious
- QML itself
  - QML lacks certain abstractions
  - Data-Binding uses QVariant, loss of type-safety

Futher outlook

- Thanks for your interest
- We are looking for companies and developers with similar QML desktop experiences
- Talk tomorrow, 11:30 in Moskau B:
  - SoDeclarative – a declarative wrapper for Coin3D

Sources

- QObjectListModel
  https://bitbucket.org/helmuts/qobjectlistmodel/
- DragNDrop
  https://bitbucket.org/gregschlom/qml-drag-drop/